BARNETT

Col. 9, line 35
Event log - register system events

Col. 10, line 6
value field 218
"wherein alerts are sent through Bonrell - network congestion "automatic alert notification"

SYSTEM FOR MONITORING AND MANAGING COMPUTER RESOURCES AND APPLICATIONS ACROSS A DISTRIBUTED COMPUTING ENVIRONMENT USING AN INTELLIGENT AUTONOMOUS AGENT ARCHITECTURE

FIELD OF THE INVENTION

This invention relates generally to computer networks. More specifically, the invention relates to a method and apparatus for centrally monitoring and managing the computers, applications and other resources present in a distributed computing environment.

BACKGROUND

The data processing resources of business organizations are increasingly taking the form of a distributed computing environment in which data and processing are dispersed over a network comprising many interconnected, heterogeneous and geographically remote computers. Among the reasons for this approach are to offload non-mission-critical processing from the mainframe, to provide a pragmatic alternative to centralized corporate databases, to establish a single computing environment, to move control into the operating divisions of the company, and to avoid having a single point of failure. For example, many business entities have one client/server network installed in each regional office, in which a high-capacity computer system operates as the "server" supporting many lower-capacity "client" desktop computers. The servers in such a business entity are also commonly connected to one another by a higher-level network known as a wide area network. In this manner, users at any location within the business entity can theoretically access resources available in the company's network regardless of where the resource is located.

The flexibility gained for users with this type of arrangement comes with a price, however. It is very difficult to manage such a diverse and widely-dispersed network for many reasons. Servers installed in the wide area network are frequently not all of the same variety. One regional office may be using an IBM machine with a UNIX operating system, while another regional office may be using a DEC machine with a VMS operating system. Also, applications present on the servers throughout the network vary not only in terms of type, but also product release level within an application type. Moreover, the applications available are changed frequently by users throughout the network, and failure events in such a network are usually difficult to catch until after a failure has already occurred. Thus, a need exists for an efficient and flexible enterprise management system.

By way of background, one computer network management system was implemented in the fashion shown schematically in FIG. 1. In FIG. 1, a network management computer system 10 is coupled via network 12 to server computer system 14 and a plurality of other server computer systems. The hardware present in each of the computer systems may be of any conventional type such as is typically found on server computers in a client/server network environment. Moreover, the hardware configuration of each of the computer systems need not be the same. For example, network management computer system 10 might be built around a computer sold by International Business Machines Corporation operating with the well-known UNIX operating system, while server computer system 14 might be built around a computer sold by Digital Equipment Corporation operating with the well-known VMS operating system. The

other server computer systems in the network might be built around yet other hardware/software platforms. In addition, all of the server computers in the network might be coupled to a variety of supported client computers such as desk-top computers, workstations and other resources. It is anticipated, however, in FIG. 1 that network management computer system 10 and each of the server computer systems in the network will be equipped with some sort of CPU 16, 18, some sort of conventional input/output equipment 20, 22 such as a keyboard and a display monitor, some sort of conventional data storage device 24, 26 such as a disk or tape drive or CD ROM drive, some sort of random access memory ("RAM") 28, 29, and some sort of conventional network communication hardware 30, 32 such as an ETHERNET interface unit for physically coupling the computer system to network 12. In the system of FIG. 1, network 12 may be implemented using any conventional network protocol such as TCP/IP. In the configuration shown in FIG. 1, a manager software system 34 is stored on storage device 24 in network management computer system 10; one agent software system is installed on each of the server computer systems in the network, such as agent software system 36 shown stored on storage device 26 in server computer system 14; at least one knowledge module 38 is stored on storage device 24 in network management computer system 10; and at least one script program 40, 42 is stored on each of the storage devices 24, 26 throughout the computer network.

FIG. 2 illustrates the main components for implementing the manager software system 34 shown in the system of FIG. 1. Knowledge module parser 44 is responsible for accessing knowledge module 38 and parsing the information therein for use by knowledge database manager 46, which in turn creates and maintains a database 47 of knowledge that is more readily useable by manager software system 34 than would be the data stored in knowledge module 38. Object database manager 48 creates and maintains a database 49 representing all of the resources and applications (collectively, "objects") present on the computer network, as well as information pertaining to the state of those objects, in a form that will be readily useable by a graphical user interface module 50. Databases 47 and 49 may be stored in RAM or on a storage device such as a hard disk. Graphical user interface 50 is responsible for communicating with display driver software in order to present visual representations of objects on the display of network management computer system 10. Such representations typically take the form of icons for objects. Also, graphical user interface module 50 coordinates the representation of system windows for command menus and the display of requested or monitored data. Event manager 52 is responsible for keeping a record of various occurrences throughout the computer network, such as the occurrence of alarm conditions and their resolution, for the purpose of record keeping and management convenience. Interface 54 is for the purpose of interfacing with network management software other than the manager software system 34 and agent software system 36. For example, users of network management computer system 10 may make use of software such as Hewlett Packard Corporation's OPENVIEW product for the purpose of monitoring low-level network conditions such as broken physical connections. While using such a third-party product, the user may open a window and request information from manager software system 34, in which case interface 54 will coordinate communication between manager software system 34 and such third party product. Communications module 56 is responsible for handling all

REQUESTED OR MONITORED DATA

OR REQUESTED TO RECEIVE ALARMS?

DISPLAY

INSTANTLY REQUESTED?

DISPLAY OF REQUESTED DATA

ALARM COND. SUCH AS?

BROKEN PHYSICAL CONNECTIONS

CURRENTLY

modules are uploaded from knowledge database 75, thus making more efficient use of server resources.

Resource Monitoring and Data Reporting with Event Filtering

Event filtering allows a console interested in a particular object to be selective about the events it wants to be notified about in relation to the object.

FIG. 22 illustrates a preferred data structure for implementing event filtering. Event filter structure 320 includes four fields. Field 322 determines whether the event filter is a PASS type filter or a FAIL type filter. Field 324 may contain information identifying events by the name of an object or list of objects potential causing the event. For example, field 324 might contain an application name, instance name or parameter name. Field 326 may contain information identifying events by type, such as a state-change event or error event. Field 328 may contain information identifying events by severity level, such as alarm severity, warning severity, or simply information-level severity. Fields 324, 326 and 328 may also contain wild-card characters.

Filtering may be explained by way of example with reference to FIG. 23. FIG. 23 illustrates a sequence of two "chained" filters, filters A and B. These two filters are designed such that if all events generated were tested against filter A first and then filter B, the result would be that all events would pass through filter A, but any event of the type having an information-level severity would be screened and would not pass through filter B. In other words, all events having a severity level higher than the information level of severity would pass through both event filters and be recognized.

A preferred procedure for monitoring resources, processing events and reporting data to consoles will now be discussed in relation to FIG. 24 and 25.

As can be seen in step 330, FIG. 24 is a loop that is repeated for each resource that the agent is supposed to monitor pursuant to the default list of resources found in the configuration file during agent initialization and pursuant to the registration infomation received and stored from registering consoles as described above. In step 332, the agent checks the state of the resource. In step 334, the agent determines whether the state of the resource has changed relative to the information stored, for example, in field 276. If the state has changed, the agent continues with step 336, in which it executes the event processing routine of FIG. 25 (to be discussed below). If not, the agent continues with step 338, in which it calculates the value of the parameters associated with this resource. In step 339, the parameter values are stored in the data structure described in FIG. 14. In step 340, the agent sends the calculated parameter values to all consoles registered to receive "real-time" data for that instance and parameter. In step 342, the agent compares the value of every parameter against a threshold (usually stored in a knowledge module) to determine if a threshold-crossing event has occurred. If so, then the agent continues with step 344, in which it executes the event processing routing of FIG. 25. If not, the agent arrives at step 346 and repeats the loop by returning to step 330 if more resources are to be monitored.

FIG. 25 is a pseudo-code listing for illustrating a preferred procedure for processing events. It will be understood that the pseudo-code is shown for purposes of explanation only, and that persons of ordinary skill in the art may program the routine differently while still remaining within the scope of the invention. Each time the routine is executed, a record of the event is logged into event repository 206. Then, two nested loops are executed. The outermost loop repeats for all consoles registered in table 250. The innermost loop repeats for all event filters in the chain of filters identified by the event context information stored in field 256 of table 250 for each registered console. The effect of the routine is that events are only reported to interested consoles, and even then only to interested consoles whose event filters are satisfied by the event. This procedure provides enhanced performance by, among other things, reducing network traffic.

Event Management

FIG. 26 is a flow diagram illustrating a preferred procedure for managing events in an enterprise management system like that of FIG. 11. It will be understood that the procedure of FIG. 26 would begin after the event processing routing of FIG. 25 sent a message to a console notifying the console that an event has occurred, and after the user at the console decided to take responsibility for handling the reported event. In step 348, after the user taking responsibility for the event makes an appropriate entry at his console, the console sends a message to the agent so indicating. In step 350, the agent modifies event repository 206 accordingly. For example, the agent would modify field 238 to identify the console or the user taking responsibility for the event. The agent might also modify field 232 indicating that the status of the event is "acknowledged." In step 352, the agent uses the data structures of FIG. 17 and 18 to send a message to every console interested in this event. Such a message would include, for example, an indication of the fact that a console has taken responsibility for the event, and an indication of the identity of the responsible console. It will be understood that all other information pertinent to the event may also be sent in such a message according to procedures already described above. In step 354, the consoles receiving a message sent by the agent in step 352 modify the entries in their own event caches 212. The result of the procedure is that the information maintained in the agent's event repository is propagated throughout the network to all interested consoles, such that the event management efforts of all interested consoles will be automatically coordinated, and such that all such consoles will have access to the same, up-to-date information about the events that interest them.

Multi-Tiered Problem Management

FIG. 27A and 27B comprise a block diagram illustrating an alternative embodiment of the invention, configured to yield multi-tiered problem monitoring and management capabilities in a large-scale enterprise. In such a configuration, the network is effectively divided into two or more tiers, such as tiers 362 and 364. It is to be understood that agent 356 in the drawing represents a multitude of other similar agents operating in tier 362 in the network. Collector 358 may be a single intermediary agent, or it may be viewed for purposes of the illustration as representing a number of such intermediate agents operating between tiers of a network, such as between tiers 362 and 364. The effect of the configuration is to reduce event-related network traffic on the higher tiers of the network, and also to remove some of the load of event management from higher level consoles. In operation, collector 358 registers with agents 356 as an SNMP manager and therefore begins to receive notification of SNMP traps. Collector 358 then converts the SNMP traps